Contents

Introduction

We have completed a project called "Perl to CQL" and this project's deliverable is a set of Perl modules in CPAN called caGRID::CQL1. There is also a SVN repository for the Perl to CQL code in GForge.

This document describes how to use the caGRID::CQL1 modules to write code that retrieves data from the Cancer Bioinformatics Grid (caGRID) using Perl. caGRID is a network of data and analytical tools based on Web Services that allows researchers to share data in the fields of cancer research and public health and analyze this data. The data in caGRID describes genes and proteins, tissues, protein expression, gene expression in the form of microarray data, and clinical data. This data is provided by individual Web Services established by more than 100 scientific and clinical laboratories in the US. A major focus of the development work in caGRID has been on standardizing the vocabulary and software architecture used by these Services so that the entire data set is query-able and interoperable.

Perl and Bioinformatics

The Perl language is a very popular one in current bioinformatics practice. Perl has been used to create and maintain important public databases such as Ensembl and is the language behind GBrowse, the world's most popular genome browser. Perl is also the scripting language of choice for most biologists and bioinformaticists in their own labs, in part due to the availability of the open source software like the Bioperl toolkit and the community actively supporting it.

Opening up caGRID to Perl users

These Perl users would benefit if they had access to the tremendous amount of high quality scientific data in caGRID. The caGRID project would benefit from the participation of these additional users who, in many cases, do not yet appreciate the full capabilities of caGRID.

The perceived barrier to these users is the sophistication and complexity of caGRID itself. The system is well-documented but it introduces unfamiliar and daunting concepts to many scientists who are not trained programmers. Many practicing biologists do not have the background required for complex installations, are unfamiliar with the nomenclature used by software developers, and will not read technical *documentation*. These researchers are ideally addressed at their level of familiarity, using their own practices. For the Perl users who are molecular biologists this means installation of the relevant Perl modules in a semi-automated fashion using CPAN (on Windows, Mac OS, or Unix), followed by the coding of scripts with the aid of some explicit *HOW TO* documentation.

The purpose of caGRID::CQL1

The caGRID::CQL1 package has been written to satisfy this Perl user, the biologist, and also the Perl developer. Though appearing as a single set of Perl modules it actually attempts to address two related needs. The first need is for modules that the biologist would use, that are dedicated to a certain type of query and are simple and accessible. The second need is for a set of Perl modules that Perl software developers would use to make new Perl modules. Essentially caGRID::CQL1 contains code to construct validated CQL queries and send them to caGRID Web Services, and these are the methods that the developers would use to make their own modules.

Perl Coding

This document assumes some familiarity with Perl coding and Perl module creation, specifically:

- General object-oriented coding in Perl
- Writing a module in an object-oriented style

For more information on these topics you can read:

- Perl Module Style
- Tutorial on Perl OOP

Installing caGRID::CQL1

UNIX

- Download the package from caGRID::CQL1.
- Install the module dependencies:

```
* XML::Writer
* LWP::UserAgent
* HTTP::Request
* XML::LibXML::Reader
* URI
* Net::HTTP
* LWP::Simple
```

- Type the following at a command line

```
tar -xvf caGRID-CQL1-X.X.X.tar
cd caGRID-CQL1-X.X.X
perl Makefile.PL
make
make test
sudo make install
```

Windows Installation

Please see the instructions at http://search.cpan.org/~JHI/perl-5.8.0/pod/perlmodinstall.pod.

Important: the caGRID-CQL1 modules do not require compilation.

Module Versions

caGRID::CQL1 has been tested with the following module versions:

```
 * XML::Writer 0.606, 0.602
 * LWP::UserAgent 5.833, 5.800
 * HTTP::Request 5.827
 * XML::LibXML::Reader 1.70, 1.64
 * URI 1.40
 * Net::HTTP 5.833
 * LWP::Simple 5.827
```

Brief Introduction to caGrid Query Language (CQL)

Most of the methods in caGRID::CQL1 are methods to construct a CQL query object. Once a CQL query object has been built, containing the CQL query XML, it can be validated and the CQL can be sent as a SOAP message to a specific caGRID service. What caGRID::CQL1 does not contain are parsers for the wide variety of result sets that will be retrieved from the different Services. Parsing would be the responsibility of a module dedicated to a specific set of Services.

It follows that in order to use caGRID::CQL1 you need to understand a bit about CQL. The following pages may be useful:

- http://cagrid.org/display/dataservices/CQL
- http://cagrid.org/display/dataservices/CQL+Examples

The CQL1::* modules in caGRID::CQL1 map directly to elements in CQL, so their documentation is also informative:

CQL1::Object

The Object holds the required 'name' of the CQL query, and these names describe the functional type or *target* of the query (e.g. 'edu.georgetown.pir.domain.ProteinSequence'). The *target* determines what data is returned by the query. The Object also serves to contain one of each of these child elements: Attribute, Association, and Group.

CQL1::Attribute

The Attribute is used to define restrictions or conditions on an Object. The 'name' of the Attribute defines its type or class (e.g. 'uniprotkbPrimaryAccession'). The 'value' of the Attribute is the query term (e.g. 'P05067', an identifier). The 'predicate' of the Attribute is the condition or operator (e.g. 'EQUAL_TO').

CQL1::Association

The Association is a subclass of Object and it is used to further define the scope of the query using its 'rolename' method. For example:

```
$assoc = new caGRID::CQL1::Association();
$assoc->rolename("protein")
```

This method call restricts the query to that available class.

CQL1::Group

A Group is used to define logical relationships or conditions between Objects, and must be a child of an Object. A Group must have two or more children, which may be a mixture of type Attribute, Association, or Group

For example, a Group can have an attribute called 'logicOperator' which has possible values of 'AND' or 'OR'. If you used the value 'AND' then all conditions in the Group must be true in order for the Group to evaluate as true.

CQL1::QueryModifier

The optional QueryModifier modifies the result set. For example, setting 'countOnly' to true means the result set will only contain the number of result rows. If you specify an 'attributeName' then only those attribute names and values will be returned.

Building CQL queries for a specific target using the caGRID Portal

In order to write code that builds CQL queries for a given target using caGRID::CQL1 you will find it useful to have specific CQL examples directed at that target and its Service. The caGRID Portal is an excellent tool for creating and reading CQL for a given target (see this Overview of the caGRID Portal for general documentation on the caGRID Portal).

We will build a CQL query using the caGRID Portal, and the resultant CQL will show how to write the caGRID::CQL1 code.

Create a CQL query using caGRID Portal

We will build a CQL query using the caGRID Portal user interface. This query will retrieve data about peptides from the PeptideAtlasService, given an identifier.

Begin by selecting **---Data Services** from the **Directories** drop-down menu.

Scroll through the Services by clicking the right-arrow and then select **PeptideAtlasService** for this example.

Copy the **URL** and paste it into the **Service URL** field on the right, and click **Select**.

Click on **org.systemsbiology.peptideatlas.Peptide**.

Click on **Add Criterion**.

Click on **id**, then select **EQUAL TO** and enter **7**.

Click **Update**.

Now the CQL has been constructed. You can execute this query by clicking **Submit Query**. To see the XML itself click **Export to XML**, you should see:

```
<ns1:CQLQuery xmlns:ns1="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery">
     <ns1:Target name="org.systemsbiology.peptideatlas.Peptide">
         <ns1:Attribute name="id" predicate="EQUAL_TO" value="7"/>
     </ns1:Target>
   <ns1:QueryModifier countOnly="true"/>
</ns1:CQLQuery>
```

Building CQL queries for a specific target using caGRID::CQL1

Now we have simple, example CQL that will retrieve data from PeptideAtlasService and we can write a script or module to build the same query using caGRID::CQL1. The module begins with some required declarations:

```
#!/usr/bin/perl -w

use strict;
use caGRID::CQL1::Attribute;
use caGRID::CQL1::Object;
use caGRID::CQL1::QueryModifier;
use caGRID::CQL1;
```

The next lines set the host and the full URL, the same information found at the caGRID Portal:

```
my $host = 'informatics.systemsbiology.net';
my $url = "http://$host:80/wsrf/services/cagrid/PeptideAtlasService";
```

We start to build the CQL here, by creating the `CQL::Object` and giving it a name. This name is the same as the *UML Class* in the caGRID Portal. The *name* is the value for the *Target* element. Remember that the *Target* determines what data type will be returned from the query.

```
$obj = new caGRID::CQL1::Object();
$obj->name("org.systemsbiology.peptideatlas.Peptide");
```

Now we add an attribute, or condition, to the query:

```
$attr = new caGRID::CQL1::Attribute();
$attr->name("id");
$attr->value("7");
$attr->predicate($caGRID::CQL1::Attribute::_EQUAL_TO);
```

Then, add this Attribute to the CQL1::Object:

```
$obj->attribute($attr);
```

The final step in building the query is modifying the query by declaring that we want just the *count* of the result, not the data itself:

```
$mod = new caGRID::CQL1::QueryModifier();
$mod->countOnly(1);
```

The query is prepared and it can be sent:

```
$cql = new caGRID::CQL1(-debug => 1);
$response = $cql->request($url, $obj, $mod);
```

Parsing the Response

Finally here is how one can parse the result of the above response, which is a *count*:

```
use XML::LibXML::Reader;

$reader = new XML::LibXML::Reader(string => $response);
$result = $cql->getResultCollection($reader);
$targetClass = caGRID::CQL1->getTargetClass($result);
my $count = caGRID::CQL1->getCount($result);
```

There are also caGRID::CQL1 methods to get the *count* of a successful query, get results as an array reference, and get results as a hash reference, but generally speaking caGRID::CQL1 does not parse the results of queries. Fortunately there are a number of good XML parsers available including:

- XML::Simple
- XML::Twig]
- XML::Parser

As implied by the name, XML::Simple is the easiest to use and would return the response in the form of a hash in this example:

```
use XML::Simple;

my $data = XMLin($response);
```

Validating the XML

It is possible to create invalid `CQLQuery` XML using caGRID::CQL1, particularly as the queries become more complex. You can validate some query against the CQL XSD like this:

```
use caGRID::CQL1::Validator;

my $xml = $cql->toXML($obj, $mod);
$xml = extractRequest($xml);

my $result = validateQueryXml($xml);
```

If `$result` is **1** then the query is valid. If the query is not valid then **0** is returned and the error code is printed to STDERR.

Example Module

An example module called GetPeptideFromPeptideAtlasService using code similar to the above code could be used like this in a script:

```
#!/usr/bin/perl;

use GetPeptideFromPeptideAtlasService;

my $q = GetPeptideFromPeptideAtlasService->new;

my $data = $q->GetPeptide(7);
```

Maintaining caGRID::CQL1 at CPAN

The caGRID::CQL1 code is kept in GForge:

- https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/projects/PerlToCQL/?root=net

The code in this directory can be packaged into a *tar file and uploaded to CPAN, where it will be automatically processed and indexed. This upload will create a new version of caGRID-CQL1.

In order to upload to CPAN you must first obtain a PAUSE account:

- https://pause.perl.org/pause/authenquery?ACTION=request_id

Certain files in the repository need to be checked before the upload to CPAN:

- Edit lib/caGRID/CQL1.pm to add the new version number
- If you have added, removed, or renamed a file make sure to edit MANIFEST
- If you have added or removed a Perl module dependency make sure to edit Makefile.PL
- If you have added, removed, or renamed a top-level file then edit make-tar.sh

A script called `make-tar.sh` is provided that will make the uploadable *tar file for you. Do something like:

```
>cd CPAN-uploads
>./make-tar.sh 1.0.1
```

Where "1.0.1" is the new version number.

Then log in to PAUSE and upload the file. Wait a few hours, and you will be informed by email when the new files are viewable in CPAN.

Appendix

This code can also be found in the Perl To CQL package.

GetPeptideFromPeptideAtlasService

An example of a simple Perl module that can be used to retrieve data about peptides from the `PeptideAtlasService`.

```
package GetPeptideFromPeptideAtlasService;
```

```
use strict;
use caGRID::CQL1::Attribute;
use caGRID::CQL1::Object;
use caGRID::CQL1::QueryModifier;
use caGRID::CQL1;
use XML::Simple;

use constant HOST => 'informatics.systemsbiology.net';
use constant URL  => 'http://' . HOST . ':80/wsrf/services/cagrid/PeptideAtlasS
use constant NAME => 'org.systemsbiology.peptideatlas.Peptide';

sub new {
        my $that = shift;
        my $class = ref($that) || $that;

        my $self = {@_};

        bless( $self, $class );
        return $self;
}

sub GetPeptide {
        my ($self,$id) = @_;

        die "No identifier" if ( ! $id );

        my $obj = new caGRID::CQL1::Object();
        $obj->name(NAME);

        my $attr = new caGRID::CQL1::Attribute();
        $attr->name("id");
        $attr->value($id);
        $attr->predicate($caGRID::CQL1::Attribute::_EQUAL_TO);

        $obj->attribute($attr);

        my $mod = new caGRID::CQL1::QueryModifier();
        $mod->countOnly(0);

        my $cql = new caGRID::CQL1(-debug => 1);
        my $response = $cql->request(URL, $obj, $mod);

        my $data = XMLin($response);

        $data;
}

1;
```

In progress

- NCIACoreService at http://imaging.nci.nih.gov:80/wsrf/services/cagrid/NCIACoreService (Can use gov.nih.nci.ncia.domain.ClinicalTrialSite)
- caArraySvc at NCICB http://array.nci.nih.gov:80/wsrf/services/cagrid/CaArraySvc
- CaBIO40GridSvc at http://cabiogrid42.nci.nih.gov:80/wsrf/services/cagrid/CaBIO42GridSvc